GRAPH THEORY Q & A

I think I did not understand very well isomorphic graphs; especially, I don't get how we can intuitively conceptualize these?

First let's make a rigorous definition

Definition 0.1. *Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be graphs. A graph isomorphism is a bijection $\phi : V_1 \to V_2$ such that $u_1v_1 \in E_1$ if and only if $\phi(u_1)\phi(v_1) \in E_2$.*

First, the idea of a graph is a set together with a binary relation. The set is the set of vertices V and the binary relation on the elements of V is determined by the set of edges E. We can think that two elements u, v of V are related if and only if there is an edge $uv \in E$.

Two finite sets are isomorphic whenever there is a bijection between the sets, and this is equivalent to the sets being of the same size. Sets do not have much structure to preserve! When we have the extra structure on a set, like a binary relation (or in the case of groups, multiplication etc) we want the notion of isomorphism to *preserve* this extra structure, which is what is required in the definition above.

With this intuition in mind, can you adapt the definition of graph isomorphism above to get a definition of an oriented graph isomorphism?

A good exercise to help understand graph isomorphisms is to ask yourself how would you check if two graphs G_1 and G_2 are isomorphic? (I am also bringing this up because the graph isomorphism problem is in NP by so far no known polynomial time algorithm is known, so it serves as a warm up to the questions below).

First you need to determine if $|V_1| = |V_2|$. If not, then there cannot be a bijection between the vertex sets and the graphs cannot be isomorphic. If the vertex sets are of the same size, we would in principle need to run through the possible bijections between V_1 and V_2 until we find one that sends edges of G_1 to edges of G_2. If there is no such bijection the graphs are not isomorphic. There are $n!$ bijections between the vertex sets if $n = |V_1| = |V_2|$.

Of course there could be some short cuts. If the degree sequences of the graphs are different the graphs cannot be isomorphic. Since if $\phi(v_1) = v_2$ for some graph isomorphism ϕ then $d(v_1) = d(v_2)$. To prove this use ϕ to construct a bijection between the sets $\{u_1 \mid u_1v_1 \in E_1\}$ and $\{u_2 \mid u_2v_2 \in E_2\}$. If the degree sequences of the two graphs are the same, when running through all bijections between V_1 and V_2 we can restrict to those which satisify $d(v_1) = d(\phi(v_1))$. If the graphs are both k-regular though, there are no such short cuts. The above approach to determining whether two graphs are isomorphic leads to an algorithm which is not polynomial in the number of vertices, it is of complexity at least $O(n!)$. After some back and

forth, the best known algorithm for the graph isomorphism problem is quasi-polynomial of order $2^{O(\log n)^c}$ for some c (we did not study this, so consider it trivia).

On the other hand if you have two graphs and I give you a candidate for an isomorphism $\phi : G_1 \to G_2$, it is rather fast to check if the two graphs are isomorphic. You can write down the adjacency matrices for the two graphs with rows and columns of the matrices indexed according to the isomorphism. Then ϕ is an isomorphism if and only if the two matrices are identical. Hence a solution can be verified in polynomial time and the graph isomorphism problem is NP. It is not known whether this problem is NP complete (see more below).

When we made the link between the connected components of a graph G and $Null(B_G^T)$ with B_G^T being the transpose of the adjacency matrix. I think I did not understand very well what we did in the proof; could you give some conceptual hints so that I can better see what we did please?

Recall that the adjacency matrix B_G of a graph is a $|V| \times |E|$ sized matrix of 0's and 1's with $b_{ie} = 0$ if i is not a vertex of an edge e and $b_{ie} = 1$ if i is a vertex in the edge e. When we used linear algebra and maps to determine the number of connected components we needed to use the adjacency matrix of the oriented graph coming from *any* choice of orientation on the edges of G. I will let $\overrightarrow{G}$ denote a fixed oriented graph coming from an arbitrary fixed orientation of the edges of G. What we do below will work for no matter which orientation you choose!

The adjacency matrix of an oriented graph is the $|V| \times |E|$ matrix $B_{\overrightarrow{G}}$ consisting of 0's 1's and -1's. The entries are given by:

If i is the start vertex of an edge e we set $b_{ie} = -1$.

If i is the end vertex of an edge e we set $b_{ie} = 1$.

If i is not a vertex of an edge e we set $b_{ie} = 0$.

A matrix B of size $n \times k$ gives a map between the vector spaces $B : \mathbb{R}^k \to \mathbb{R}^n$. The map is simply $B(\mathbf{x}) = B\mathbf{x} \in \mathbb{R}^n$ for $\mathbf{x} \in \mathbb{R}^k$. When we take the matrix $B_{\overrightarrow{G}}^T$ we get a map from $\mathbb{R}^{|V|} \to \mathbb{R}^{|E|}$. How can we intuitively think of this map and the vector spaces $\mathbb{R}^{|V|}$ and $\mathbb{R}^{|E|}$ in terms of the graph?

A vector in $\mathbb{R}^{|V|}$ can be thought of as an assignment of a real number to each vertex of the graph. You can think of this as a sort of weight. Equivalently, and element $(y_1, \ldots, y_n) \in \mathbb{R}^{|V|}$ can be thought of as a function $f : V \to \mathbb{R}$ defined by $f(v_i) = y_i$. Analogously, an element $(x_1, \ldots, x_q) \in \mathbb{R}^{|E|}$ can be thought of as a function $g : E \to \mathbb{R}$, defined by $g(e_i) = x_i$. Alternatively, you can think of x_i as the assignment of a weight (real number) to the edge e_i. With this interpretation in mind, the matrix $B_{\overrightarrow{G}}^T$ gives us a way of transforming a function on the vertices of G to functions on the edges of G. This "transformation" depends on the way we chose to orient the edges. The map $B_{\overrightarrow{G}}^T : \mathbb{R}^{|V|} \to \mathbb{R}^{|E|}$, given by matrix vector multiplication can then be written down as:

$$B^T_{\vec{G}}((y_1, \ldots, y_n)) = (x_1, \ldots, x_q)$$

where $x_i = y_{i_j} - y_{i_k}$ if the oriented edge e_i has start vertex u_{i_k} and end vertex u_{i_j}. In other words, if you have a weights on the vertices, you can assign weights to edges of G by prescribing the weight of an edge to be the weight of the end vertex minus the weight of the start vertex.

We now want to find which vectors in $\mathbb{R}^{|V|}$ are sent to $\mathbf{0}$ by this map. For a $(y_1, \ldots, y_n)$ to be sent to zero under the map, the weight which it produces on each edge must be zero. In particular, for each edge we have to have $y_{i_j} = y_{i_k}$, where u_{i_k} and u_{i_j} are its vertices. Therefore, vertices which are connected by an edge must have the same values of y_i's, and by transitivity any vertices connected by a path must also have the same values of y_i's. Therefore, $(y_1, \ldots, y_n) \in Null(B^T_{\vec{G}})$ if and only if the entries of the vector y are constant on the vertices contained in the connected components of G.

For each connected component of G, we can cook up a vector in $Null(B^T_{\vec{G}})$, namely the vector that has entry 1 for any vertex in the connected component and 0's everywhere else. These vectors form a basis of $Null(B^T_{\vec{G}})$, so $\dim Null(B^T_{\vec{G}})$ is equal to the number of connected components. It is probably best to go back at this point (or even before this) and look at the examples in the PDF from the oriented graph day!

Could we go a little deeper regarding NP-completeness or could you direct me toward some videos that explain it? I tried to find such things but I found nothing very relevant and I would like to better visualize what it is as it seems a very important and interesting topic.

Angaende P, NP, og NP-komplette problemer lurte jeg pa om du kunne forklare hvordan man viser/vet at et problem er NP-komplett og ikke bare "vanlig" NP?

Let's review the main classes of problems we studied in this context. First we had P problems, these are problems for which there exists a polynomial time algorithm to solve then. Next, a problem is in NP if there exists a polynomial time algorithm to *verify* a proposed solution. The abbreviation NP is for "non-deterministic polynomial". Given a problem Π we say write $\Pi \in P$ if there is a polynomial time algorithm for Π, similarly $\Pi \in NP$ if there exists a polynomial time algorithm for verifying a potential solution to Π.

Just because we don't know a polynomial time algorithm for a fixed problem (or its verification), doesn't mean that one doesn't exist! For example, it was only in 2002 that Agrawal, Kayal, and Saxena showed that determining if a number is prime is in P. Their paper is here.

Any problem that is in P is also in NP, since to verify a possible solution, you could use the polynomial time algorithm to first solve the problem and

then compare your solution to the one you had to check. If we think of P and NP as collections of problems then $P \subseteq NP$.

The question of whether or not $P = NP$ is considered one of the hardest problems in mathematics and is certainly one of the most important problems in complexity theory. For example, if $P = NP$ it would mean that there is a polynomial time algorithm for factoring integers since verifying a potential factorisation requires only multiplication and multiplication is polynomial complexity. If we think about RSA and other encryption protocols, we see that the question has real world implications to data security.

Then there were the NP complete problems. These are the problems that are the hardest among the NP problems. More precisely, a problem Π is NP complete if $\Pi \in P$ implies $\Pi' \in P$ for all other $\Pi' \in NP$. So if any NP-complete problem is in P, then we would have $P = NP$. Therefore these problems are the ones of significant interest.

The way to show that an NP problem Π is in fact in the NP-complete club, is to find a known NP complete problem Π' and show that if you can solve Π in polynomial time you can solve Π' in polynomial time. In a sense, reduce Π' to Π in polynomial time. This approach clearly only works if you already have some NP-complete problems laying around.

The first ever problem that was shown to be NP-complete is the Boolean satisfiability problem (SAT). This problem asks, given a Boolean expression can you find a substitution of all of the variables as TRUE/FALSE so that the expression evaluates to be true? The Cook-Levin Theorem from the 1970's proves that the Boolean satisfiability problem is NP complete. The idea of the proof is to show that from any problem in NP can be reduced to a Boolean satisfiability problem.

Known NP complete problems that we have seen sides of in the course are the traveling salesman problem, Hamiltonian path problem, the subgraph isomorphism problem (the graph isomorphism problem is in NP, but not known to be NP complete), graph coloring problem, and determining a vertex cover.

K Shaw, Department of Mathematics, University of Oslo, P.O box 1053, Blindern, 0316, Oslo, Norway krisshaw@math.uio.no